

# HATA KONTROL MEKANİZMALARI

## ▼ Nedir?

hatalarımızı yakalamak

## ▼ Derleyici SözDizimi Hataları -Syntax Error -

- Derleme sırasında ve gelişmiş editörler derlemeye gerek kalmaksızın direkt karşılaşıyoruz.

## ▼ Runtime Hataları

- Syntax da bir sorun yok
- kod semantiği kusursuz
- ama çalışma zamanında hata veriyorsa işte bu runtime errordur. Çalışma zamanı hatasıdır.
- Programın işletim sistemi tarafınan kesilmesidir.
- Programı geliştirirken hata ile karşılaşmam olası bir durum varsa bu durum önceden tespit edilip daha anlaşılabilir bir şekilde düzenlenmelidir.
- Try catch yapısı kullanılır.
- Hangi durumlarda runtime hatası alabiliriz.
  - Olmayan dosya açmaya, yazmaya, okumaya vs çalıştığımızda
  - Olmayan değerler üzerinde işlem yapmaya çalıştığımızda
  - Uygun olmayan formatlarla çalıştığımızda
  - Veritabanı bağlantısını kopması durumunda

## ▼ Try-catch

- Olası hataları kontrol etmemizi sağlar.

Çalışma zamanında alınan olası hataları kontrol etmemizi, karalamamızı, manipüle etmemizi sağlayan bir yapılandırma.

try-catch yapılması, uygulama sürecinde yaşanan olası hatayı kullanıcıya hissettirmeksizin farklı bir duruma ya da oluşan bir mesaj gibi göstermemizi sağlayan ve bunun yanında patlamaya/hataya dair bizlere bilgi sunan ve böylece bu bilgiler eşliğinde kayıtlar/log oluşturmamızı sağlayan bir programatik yapılandırma.

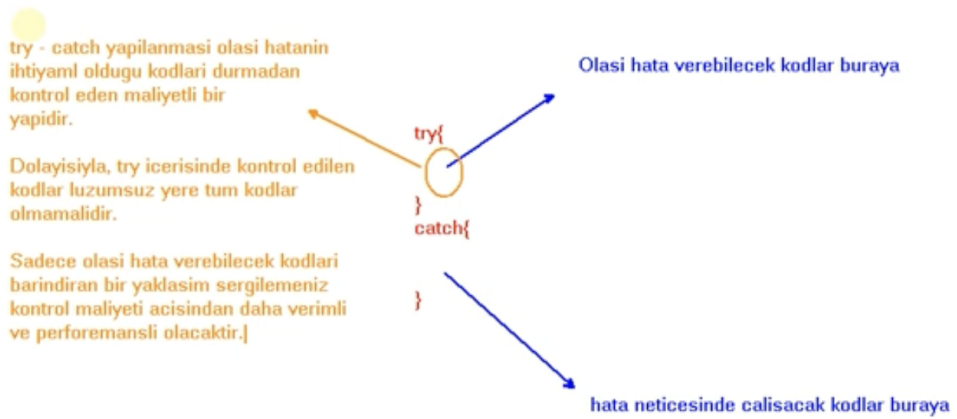
**AMAC**

1. Kullanıcıya alınan hatayı hissettirmemek.
2. Alınan hatanın nedene dair kullanıcıyı bilgilendirmek.
3. İşletim sistemleri aykırı durum yaşandığında uygulamayı sonlandırmak isterler ve sonlandırılır.

try catch yapılması ile alınan hataya dair bir manipülasyon gerçekleştiriliyor ve uygulamanın kapanmadan devam edilmesi sağlanabiliyor....

**PROTOTİP**

```
try
{
    //Olası çalışma zamanı hatalarını verebilecek kodları buraya yazıyoruz.....
}
catch
{
    //try içerisinde olası hata söz konusuysa kodun akışı orada kesilecek ve akış catch bloğundan devam edecektir.
}
```



```
irinci sayıyı giriniz.");
le.ReadLine());
. . . . .
```

- Try catch Hata parametreleri
  - catch yapısı içerisinde hatanın türüne göre farklı hata mesajları fırlatılıyor.

```
try
{
    int s1 = 0, s2 = 10;
    int a = s2 / s1;
}
catch (Exception ex)
{
}
#endregion
```

bu parametre catch bloğuna yazılmak/tanımlanmak zorunda değildir....  
eğer ki tanımlama yapılsa hataya dair bilgi alabiliriz.... yok eğer tanımlama yapılmazsa hata neticesinde catch çalışacak lakin bilgi alamayacağız...

○

```

try
{
    int s1 = 0, s2 = 10;
    int a = s2 / s1;
}
catch (Exception ex)
{
    Console.WriteLine("Mesaj : " + ex.Message);
}
#endregion

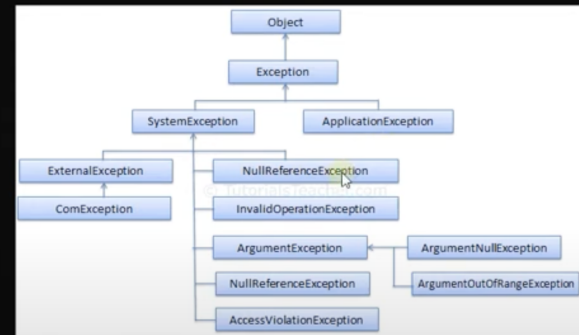
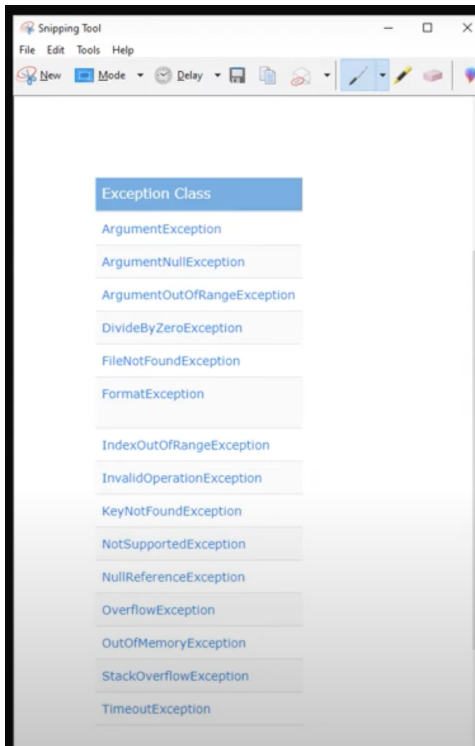
```

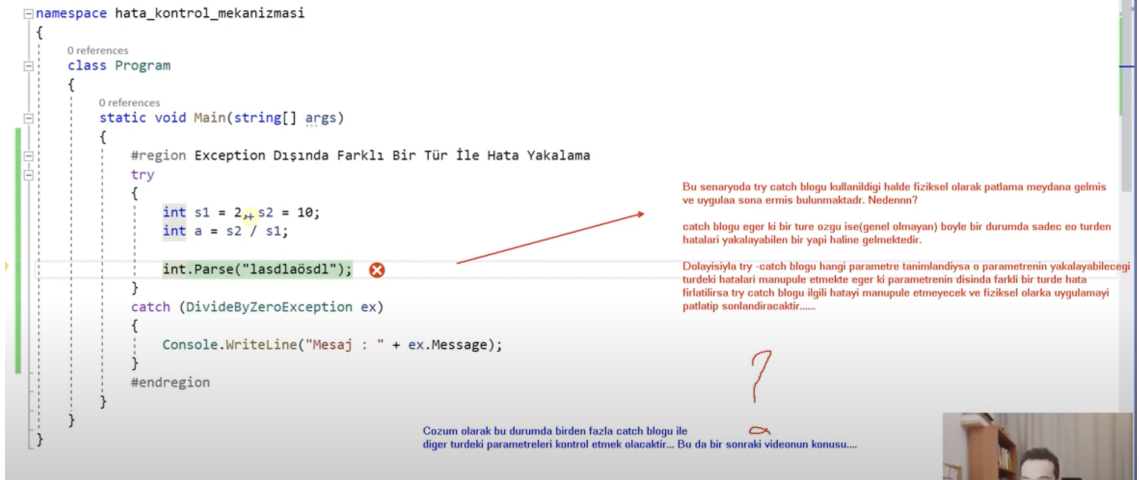
Select Microsoft Visual Studio Debug Console

Mesaj : Attempted to divide by zero.

C:\Users\Gencay\Source\Repos\Application\hata\_kontrol\_mek...  
process 19000) exited with code 0.  
To automatically close the console when debugging stops, e...  
le when debugging stops.  
Press any key to close this window . . .

- Hata türleri





- Farklı parametrelerle birden çok catch bloğu kullanılabilir. try catch catch catch...
- finally bloğu
  - hata alınsa da alınmasa da çalıştırılması gereken kodları yazdığımız bloktur. try catch lerden sonra finally {} şeklinde tanımlanır.
- when yapısı
  - try catch bloklarına when keywordü gelerek daha çeşitli şartlar sağlayarak daha detaylı araştırmalar yapabiliriz.
  - try{} catch (Ex ex ) when(x==1) {} ise şeklinde yazılır prototipi

#### ▼ Mantıksal Hatalar

- Günlük hayattaki karşılığı bug dır. Mantıksal bir sorundur. Tespiti çok zordur. Kod akışında, pc de hata yok vs.
- Bazen tek çözüm debug'dır.
- Örneğin 2 ve 5 in çarpılması istenirken yanlışlıkla 2\*6 yazıldı. Bir mantıksal hatadır.